Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота № 4

із дисципліни ««Технології розроблення програмного забезпечення»
Тема: «Вступ до паттернів проектування»

Виконав:
студент групи ІА-31:
Кунда А.П.

Перевірив:
Мягкий М.Ю

Київ 2025

Тема: Вступ до паттернів проектування.

Тема проєкту: 28. JSON Tool (ENG) (strategy, command, observer, template method,flyweight) Display JSON schema with syntax highlight. Validate JSON schema and display errors. Create user friendly table\list box\other for read and update JSON schema properties metadata (description, example, data type, format, etc.). Autosave\restore when edit, maybe history. Can check JSON value by schema (Put schema and JSON = valid\invalid, display errors). Export schema as markdown table. JSON to "flat" view.

Мета: Вивчити структуру шаблонів «Singleton», «Iterator», «Proxy», «State», «Strategy» та навчитися застосовувати їх в реалізації програмної системи.

Посилання на репозиторій з проєктом:
https://github.com/Octopus663/json-tool

Хід роботи

1. Ознайомитись з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Реалізувати один з розглянутих шаблонів за обраною темою.
4 Реалізувати не менше 3-х класів відповідно до обраної теми

За допомогою патерну «Strategy» я реалізую вимогу "Validate JSON schema and display errors"

```
<dependency>
    <groupId>com.networknt</groupId>
    <artifactId>json-schema-validator</artifactId>
    <version>1.4.0</version>
</dependency>

<dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20240303</version>
</dependency>
```

Код доданих залежностей в pom.xml

```java
package com.example.jsontool.validation;

import org.json.JSONException;
import org.json.JSONObject;
import org.json.JSONArray;
import java.util.List;
import java.util.Collections;

public class SyntaxValidation implements Validation {  no usages  new *

    @Override  no usages  new *
    public List<String> validate(String jsonText, String schemaText) {
        try {
            new JSONObject(jsonText);
        } catch (JSONException e1) {
            try {
                new JSONArray(jsonText);
            } catch (JSONException e2) {
                return List.of( e1: "Invalid JSON syntax: " + e2.getMessage());
            }
        }
        return Collections.emptyList();
    }
}
```

Код класу SyntaxValidation

```java
package com.example.jsontool.validation;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.networknt.schema.JsonSchema;
import com.networknt.schema.JsonSchemaFactory;
import com.networknt.schema.SpecVersion;
import com.networknt.schema.ValidationMessage;

import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;

public class SchemaValidation implements Validation {  no usages  new *

    private final ObjectMapper objectMapper = new ObjectMapper();  2 usages
    private final JsonSchemaFactory factory = JsonSchemaFactory.getInstance(SpecVersion.VersionFlag.V7);  1 usage

    @Override  no usages  new *
    public List<String> validate(String jsonText, String schemaText) {
        try {
            JsonNode schemaNode = objectMapper.readTree(schemaText);
            JsonSchema schema = factory.getSchema(schemaNode);

            JsonNode jsonNode = objectMapper.readTree(jsonText);
            Set<ValidationMessage> errors = schema.validate(jsonNode);
            return errors.stream()  Stream<ValidationMessage>
                    .map(ValidationMessage::getMessage)  Stream<String>
                    .collect(Collectors.toList());

        } catch (Exception e) {
            return List.of( e1: "Validation failed due to an error: " + e.getMessage());
        }
    }
```

Код класу SchemaValidation

```java
import com.example.jsontool.service.DocumentService;
import com.example.jsontool.service.ValidationService;
import com.example.jsontool.validation.SchemaValidation;
import com.example.jsontool.validation.SyntaxValidation;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import java.util.List;

@PostMapping("/validate") new *
@ResponseBody
public Map<String, List<String>> validateDocument(
        @RequestParam("jsonText") String jsonText,
        @RequestParam("schemaText") String schemaText) {

    List<String> errors;
    if (schemaText == null || schemaText.trim().isEmpty()) {
        errors = validationService.executeValidation(new SyntaxValidation(), jsonText, schemaText: null);
    } else {
        errors = validationService.executeValidation(new SchemaValidation(), jsonText, schemaText);
    }
    return Map.of( k1: "errors", errors);
```

Код класу DocumentController

```javascript
<script th:src="@{/js/validation.js}" th:inline="javascript">
</script>
<script>
  document.getElementById('validate-btn').addEventListener('click', function() {
    let jsonText = document.getElementById('content').value;
    let schemaText = document.getElementById('schema').value;
    let resultsDiv = document.getElementById('validation-results');

    let formData = new URLSearchParams();
    formData.append('jsonText', jsonText);
    formData.append('schemaText', schemaText);

    fetch('/documents/validate', {
      method: 'POST',
      body: formData
    })
            .then(response => response.json())
            .then(data => {
              resultsDiv.innerHTML = ''; // Очищуємо старі результати

              if (data.errors && data.errors.length > 0) {
                let errorList = data.errors.map(err => `<li>${err}</li>`).join('');
                resultsDiv.innerHTML = `<div class="errors"><strong>Валідація не пройдена:</strong><ul>${errorList}</ul></div>`;
              } else {
                resultsDiv.innerHTML = `<div class="success"><strong>Валідація успішна!</strong></div>`;
              }
            })
            .catch(error => {
              console.error('Error:', error);
              resultsDiv.innerHTML = `<div class="errors"><strong>Сталася помилка запиту.</strong></div>`;
            });
  });
</script>
```

Код у файлі create-document для валідування json

Результат

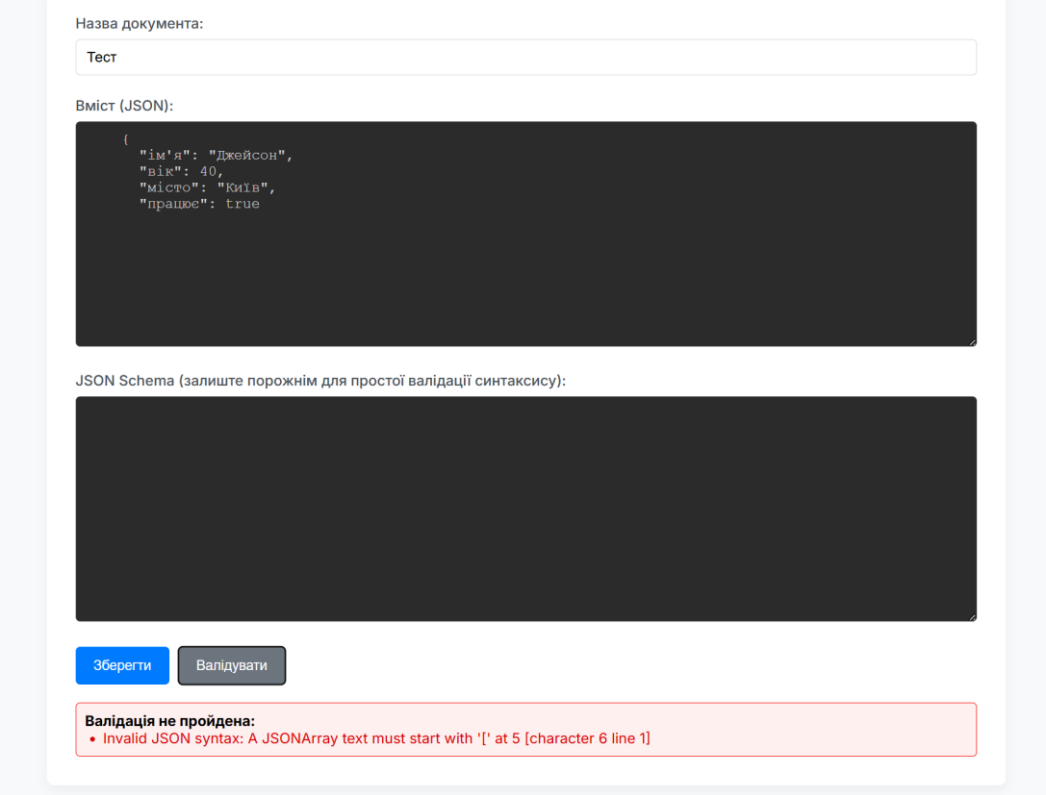


Рисунок 1 – Помилка синтаксису

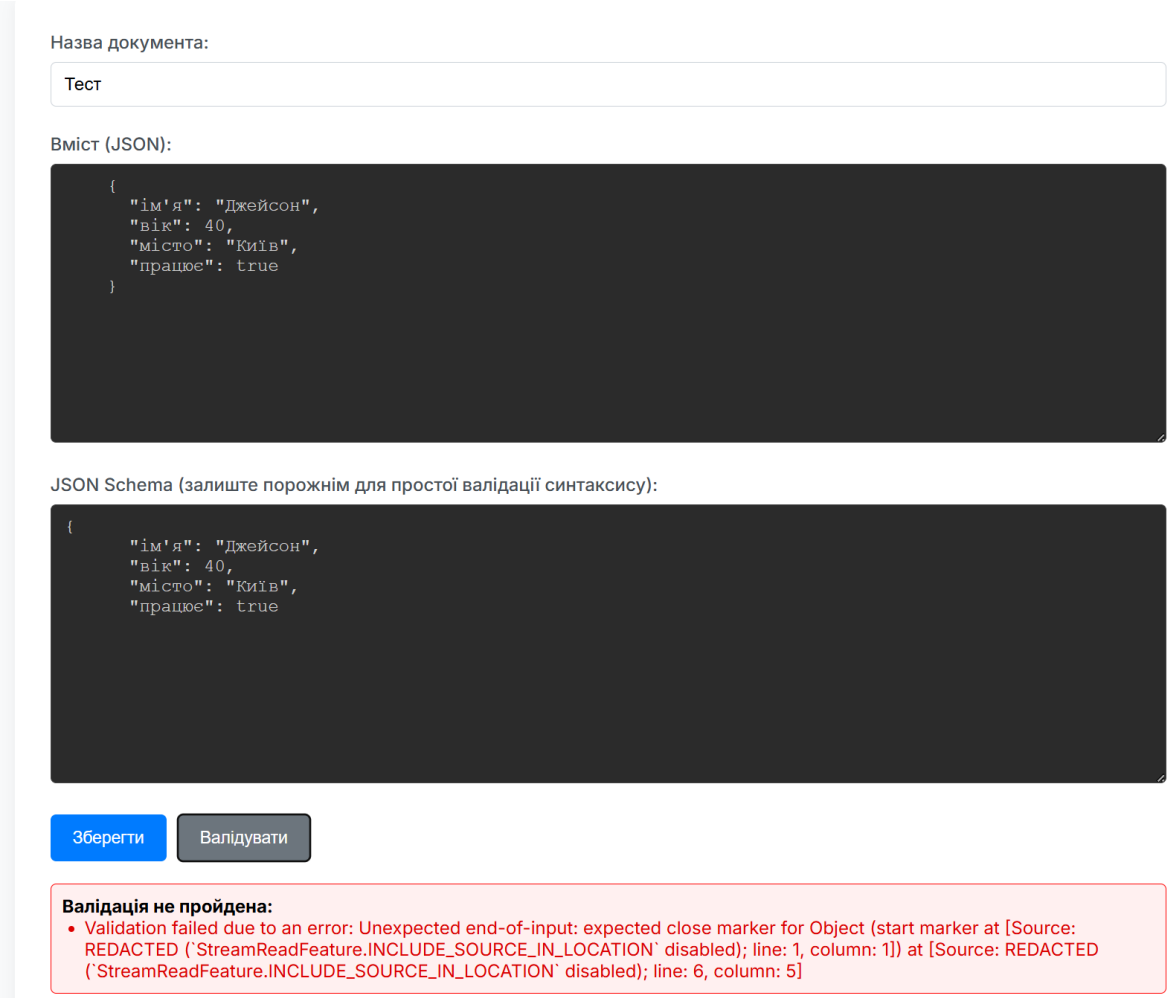


Рисунок 2 – Помилка схеми

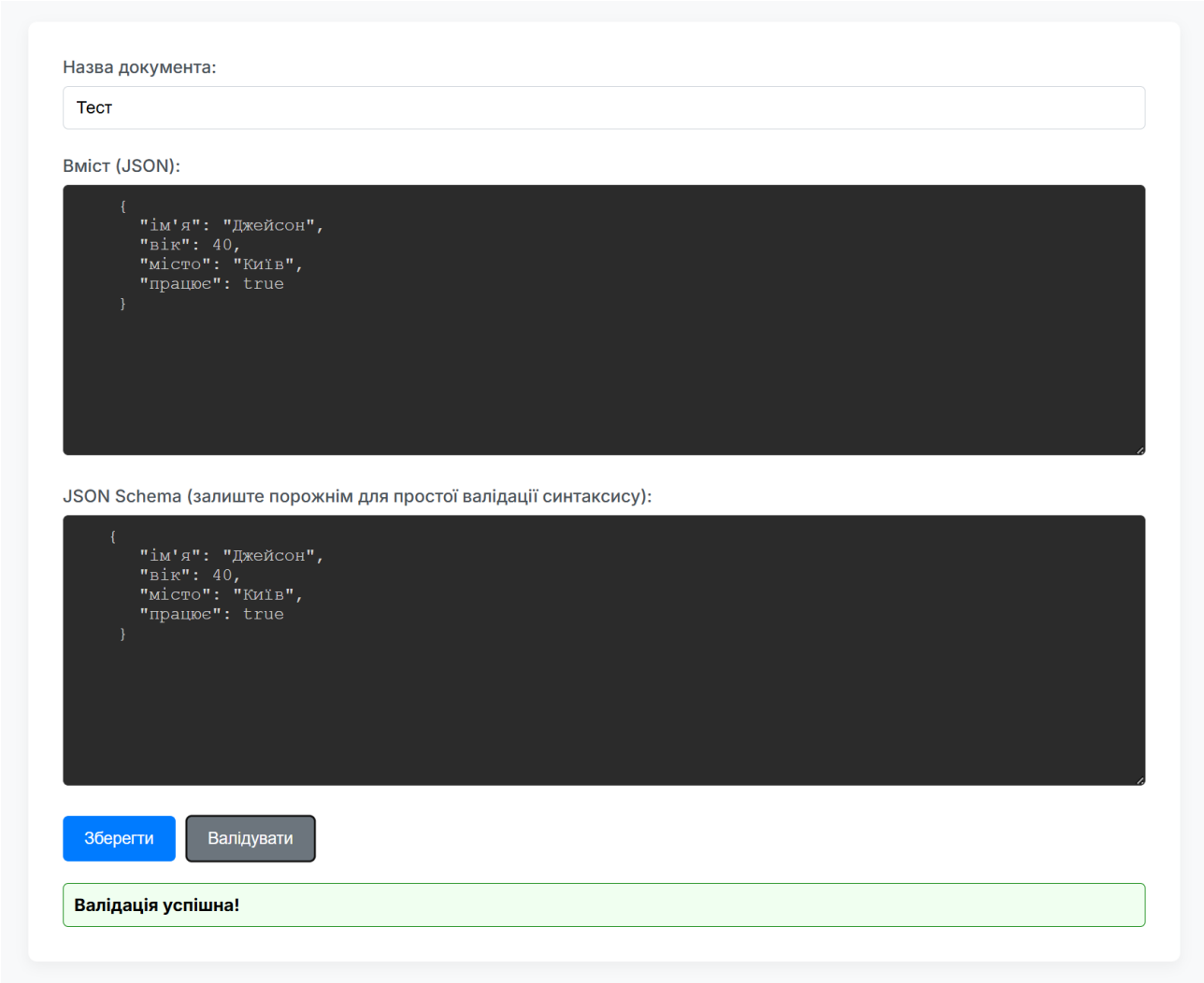


Рисунок 3 – Успіх валідації

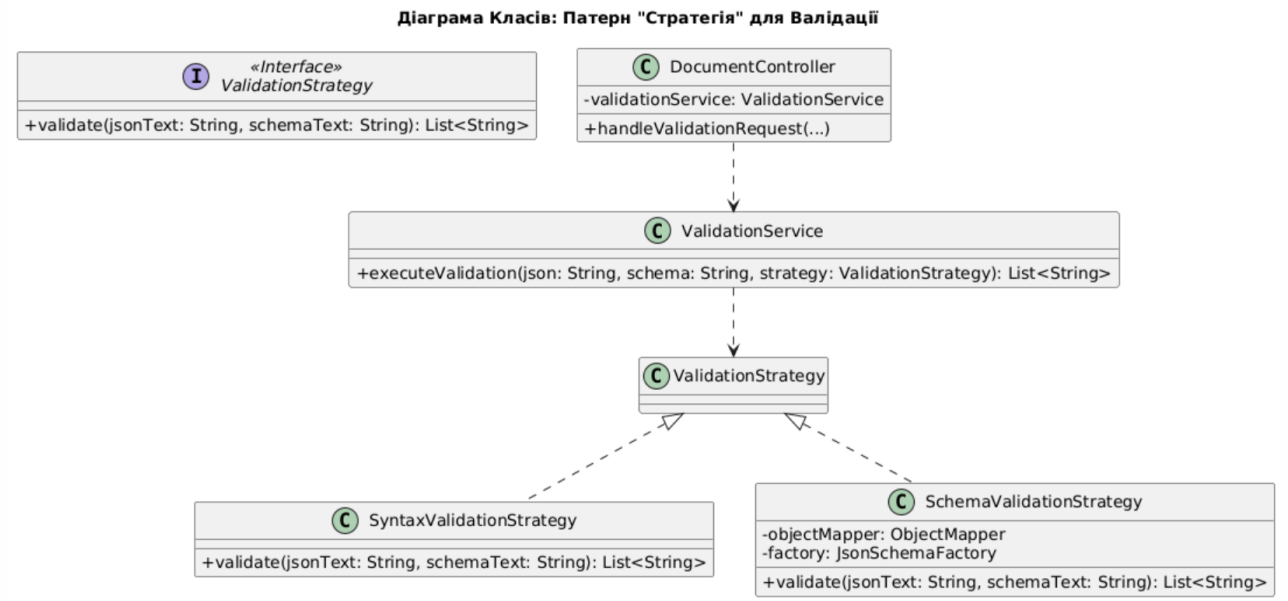


Рисунок 4 - Структура патерну Стратегія

5. Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону

Контрольні питання

1. Що таке шаблон проєктування?

Шаблон проєктування (Design Pattern) — це готове, перевірене часом, багаторазове рішення типової проблеми, що виникає при проєктуванні програмного забезпечення. Це не готовий фрагмент коду, а скоріше **концепція або опис** взаємодії об'єктів та класів для вирішення певного завдання.

2. Навіщо використовувати шаблони проєктування?

1. **Спільна мова:** Вони надають розробникам спільний словник для обговорення архітектурних рішень.
2. **Економія часу:** Не потрібно "винаходити велосипед" для вирішення стандартних проблем.
3. **Якість коду:** Код, побудований на шаблонах, зазвичай є більш гнучким, зрозумілим та легким у підтримці.

3. Яке призначення шаблону «Стратегія»?

Шаблон «Стратегія» (Strategy) дозволяє визначити **сімейство схожих алгоритмів**, "запакувати" кожен з них в окремий клас і зробити їх **взаємозамінними**. Це дозволяє клієнту обирати потрібний алгоритм під час виконання програми, не змінюючи код самого клієнта.

(У вашому випадку, ValidationStrategy — це інтерфейс, а SyntaxValidationStrategy та SchemaValidationStrategy — це конкретні, взаємозамінні алгоритми).

4. Нарисуйте структуру шаблону «Стратегія».

5. Які класи входять в шаблон «Стратегія», та яка між ними взаємодія?

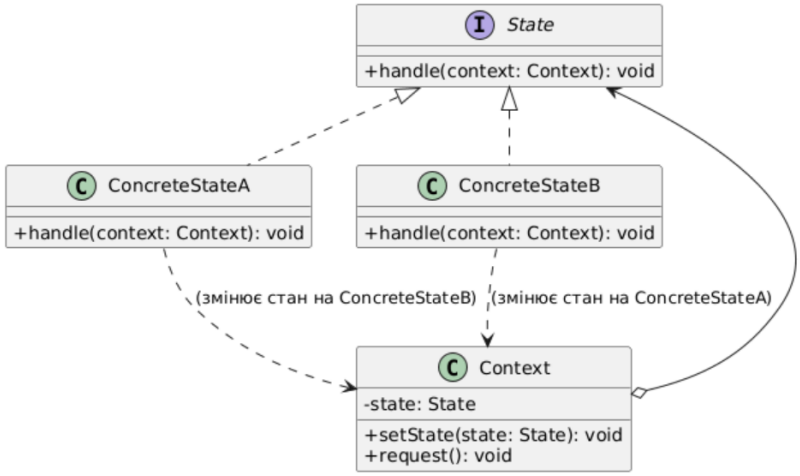
1. **Strategy (Стратегія):** Це **інтерфейс**, який оголошує загальний метод для всіх алгоритмів (наприклад, validate(...)).
2. **ConcreteStrategy (Конкретна стратегія):** Це **класи**, що реалізують інтерфейс Strategy. Кожен клас надає свою унікальну реалізацію алгоритму (наприклад, SyntaxValidationStrategy, SchemaValidationStrategy).
3. **Context (Контекст):** Це **клас**, який використовує стратегію. Він містить посилання на об'єкт Strategy і викликає його метод. Контекст не знає, *який саме* алгоритм він виконує; він просто знає, що алгоритм реалізує потрібний інтерфейс.

Взаємодія: Клієнт (наприклад, ваш DocumentController) створює об'єкт Context (ваш ValidationService) і "передає" йому потрібну ConcreteStrategy. Коли клієнт викликає метод у Context (наприклад, executeValidation), Context делегує цей виклик методу execute обраного об'єкта Strategy.

6. Яке призначення шаблону «Стан»?

Шаблон «Стан» (State) дозволяє об'єкту **змінювати свою поведінку**, коли змінюється його **внутрішній стан**. Ззовні це виглядає так, ніби об'єкт змінив свій клас.

7. Нарисуйте структуру шаблону «Стан».



8. Які класи входять в шаблон «Стан», та яка між ними взаємодія?

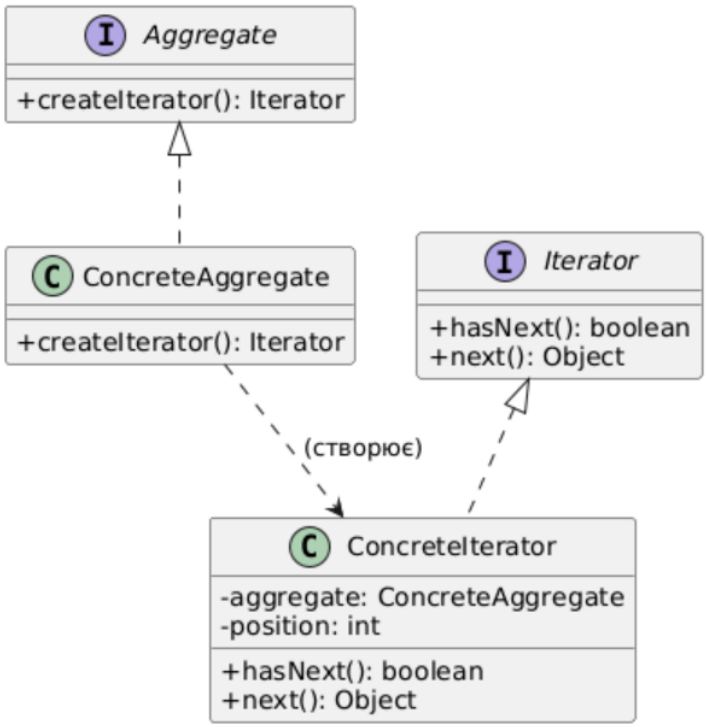
1. **Context (Контекст):** Об'єкт, поведінка якого змінюється. Він зберігає посилання на свій поточний стан.
2. **State (Стан):** Інтерфейс, що визначає методи, поведінка яких залежить від стану.
3. **ConcreteState (Конкретний стан):** Класи, що реалізують поведінку для кожного конкретного стану.

Взаємодія: Context делегує всі запити своєму поточному об'єкту State. У відповідь на запит (або іншу подію), ConcreteState може виконати дію і, що важливо, "перемкнути" стан Context на інший ConcreteState.

9. Яке призначення шаблону «Ітератор»?

Шаблон «Ітератор» (Iterator) надає **уніфікований спосіб** послідовного доступу до елементів колекції (наприклад, списку чи масиву), **не розкриваючи її внутрішньої структури** (чи це ArrayList, чи HashSet тощо).

10. Нарисуйте структуру шаблону «Ітератор».



11. Які класи входять в шаблон «Ітератор», та яка між ними взаємодія?

1. **Iterator (Ітератор):** Інтерфейс, що визначає методи для обходу колекції (hasNext(), next()).
2. **ConcreteIterator (Конкретний ітератор):** Реалізація Iterator, яка знає, як обходити конкретну колекцію і відстежує поточну позицію.
3. **Aggregate (Колекція):** Інтерфейс, що визначає метод для отримання ітератора (createIterator()).
4. **ConcreteAggregate (Конкретна колекція):** Клас, що реалізує інтерфейс Aggregate і повертає екземпляр ConcreteIterator, пов'язаний з цією колекцією.

Взаємодія: Клієнт просить у ConcreteAggregate об'єкт Iterator. ConcreteAggregate створює ConcreteIterator і повертає його клієнту. Клієнт використовує лише методи Iterator (hasNext(), next()) для обходу, не знаючи, як влаштована ConcreteAggregate.

12. В чому полягає ідея шаблона «Одинак»?

Шаблон «Одинак» (Singleton) гарантує, що клас має **тільки один-єдиний екземпляр** (об'єкт) у всій програмі, і надає **глобальну точку доступу** до цього екземпляра. Це досягається шляхом приховування конструктора (робить його private) і надання статичного методу getInstance(), який створює об'єкт при першому виклику і повертає його при всіх наступних.

13. Чому шаблон «Одинак» вважають «анти-шаблоном»?

Його вважають "анти-шаблоном" (поганою практикою) у сучасній розробці з кількох причин:

1. **Глобальний стан:** Він створює глобальну змінну, що ускладнює відстеження залежностей і робить код непередбачуваним.

2. **Складність тестування:** Неможливо легко "підмінити" (mock) одинака для юніт-тестів, що робить тестування складним або неможливим.
3. **Порушення принципу єдиної відповідальності (SRP):** Клас-одинак відповідає і за свою бізнес-логіку, і за контроль над власним життєвим циклом.

У Spring Boot, наприклад, замість Singleton використовується **Dependency Injection**, який дає ті ж переваги (один екземпляр на весь додаток), але не має його недоліків.

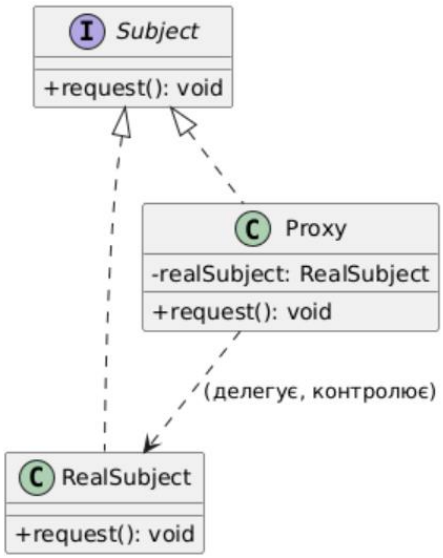
14. Яке призначення шаблону «Проксі»?

Шаблон «Проксі» (Proxy) або «Замісник» надає **об'єкт-посередник** (сурогат), який **контролює доступ** до іншого, "справжнього" об'єкта. Клієнт працює з проксі, ніби це і є справжній об'єкт.

Це використовується для:

- **Ледачої ініціалізації** (створення "важкого" об'єкта тільки тоді, коли він справді потрібен).
- **Контролю доступу** (перевірка прав користувача перед виконанням методу).
- **Логування** або **кешування** запитів.

15. Нарисуйте структуру шаблону «Проксі».



16. Які класи входять в шаблон «Проксі», та яка між ними взаємодія?

1. **Subject (Суб'єкт):** Це **інтерфейс**, який реалізують і проксі, і справжній об'єкт.
2. **RealSubject (Справжній суб'єкт):** "Важкий" або "захищений" клас, який виконує основну роботу.
3. **Proxy (Замісник):** Клас, який також реалізує Subject. Він містить посилання на RealSubject.

Взаємодія: Клієнт завжди спілкується тільки з Proxy. Коли клієнт викликає метод request() у Proxy, Proxy виконує свою додаткову логіку (наприклад, перевіряє права доступу). Якщо все гаразд, Proxy **делегує** (перенаправляє) цей виклик справжньому об'єкту RealSubject